Computers & Operations Research 51 (2014) 123–129

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Optimization procedures for the bipartite unconstrained 0-1 quadratic programming problem

Abraham Duarte [a], Manuel Laguna [b], Rafael Martí [c], Jesús Sánchez-Oro [a]

[a] Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles, Spain
[b] Leeds School of Business, University of Colorado at Boulder, Boulder, CO, USA
[c] Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Valencia, Spain

ARTICLE INFO

Available online 27 May 2014

Keywords:
Quadratic programming
Branch and bound
Heuristic search
Tabu search
Iterated local search

ABSTRACT

The bipartite unconstrained 0-1 quadratic programming problem (BQP) is a difficult combinatorial problem defined on a complete graph that consists of selecting a subgraph that maximizes the sum of the weights associated with the chosen vertices and the edges that connect them. The problem has appeared under several different names in the literature, including maximum weight induced subgraph, maximum weight biclique, matrix factorization and maximum cut on bipartite graphs. There are only two unpublished works (technical reports) where heuristic approaches are tested on BQP instances. Our goal is to combine straightforward search elements to balance diversification and intensification in both exact (branch and bound) and heuristic (iterated local search) frameworks. We perform a number of experiments to test individual search components and also to create new benchmarks when comparing against the state of the art, which the proposed procedure outperforms.

1. Introduction

The bipartite unconstrained 0-1 quadratic programming problem (BQP) consists of selecting a subgraph that maximizes the sum of the weights associated with the chosen vertices and the edges that connect them. The problem is defined on a complete bipartite graph $G=(V,E)$, with $I=\{1,2,\ldots,m\}$ representing the set of vertices in the left-hand side of the graph, $J=\{1,2,\ldots,n\}$ representing the set of vertices in the right-hand side of the graph, E representing the set of edges that connect the vertices in I with the vertices in J, and $V=I\cup J$. There is a weight c_v associated with each vertex $v\in V$. There is also a weight q_{ij} that corresponds to the edge connecting vertices $i\in I$ and $j\in J$.[1] The problem consists of selecting $S\subseteq V$ such that the following function is maximized

$$f(S)=\sum_{v\in S}c_v+\sum_{i,j\in S}q_{ij}$$

Fig. 1 shows an example with $I=\{a,b,c\}$, $J=\{w,x,y,z\}$ and the table of edge weights. We assume that all vertex weights are zero (i.e., $c_v=0\ \forall\ v\in V$). We point out that weights, either on the vertices or the edges, can be positive, negative or zero.

Consider a solution $S_1=\{a,w,x,z\}$. The objective function value of this solution is

$$f(S_1)=q_{(a,w)}+q_{(a,x)}+q_{(a,z)}=8-4+13=17$$

A better solution is obtained by making the following vertex selections: $S_2=\{b,c,w,y,z\}$. The objective function value of solution S_2 is

$$f(S_2)=q_{(b,w)}+q_{(b,y)}+q_{(b,z)}+q_{(c,w)}+q_{(c,y)}+q_{(c,z)}$$
$$=1-7+24-15+8+20=31$$

In this case, an increase in the number of vertices from solution S_1 to solution S_2 resulted in an increase in the objective function of 14 units (31–17=14). However, this is not necessarily true in all cases. For instance, selecting a and c on the left side and y and z on the right side results in an objective function value of 38. This solution has four vertices and is better than solution S_2 that has 5 vertices.

The BQP has been studied in the literature under different names: maximum weight induced subgraph [19], maximum weight biclique [2], matrix factorization [7] or maximum cut on bipartite graphs [1]. From the point of view of heuristic optimization, however, the BQP has been somewhat neglected. In particular, to the best of our knowledge, there exist two articles – currently available on line – that describe several heuristics

E-mail addresses: Abraham.Duarte@urjc.es (A. Duarte), laguna@colorado.edu (M. Laguna), Rafael.Marti@uv.es (R. Martí), jesus.sanchezoro@urjc.es (J. Sánchez-Oro).

[1] Throughout our descriptions, we will use i to denote vertices in the left-hand-side of the graph (i.e., vertices in I), j to denote vertices in the right-hand-side of the graph (i.e., in J), and v to denote vertices in either side.

http://dx.doi.org/10.1016/j.cor.2014.05.019
0305-0548/

Edge	Weight
(a,w)	+8
(a,x)	-4
(a,y)	-3
(a,z)	+13
(b,w)	+1
(b,x)	0
(b,y)	-7
(b,z)	+24
(c,w)	-15
(c,x)	-10
(c,y)	+8
(c,z)	+20

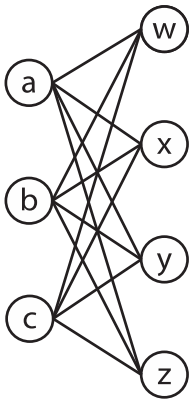

Fig. 1. BQP example.

for this problem [11]. This work develops 24 heuristics that are grouped into three families: fast-heuristics, slow-heuristics and row-merge heuristics.

In the reminder of the paper, we first introduce an exact method in the form of a branch-and-bound search. We describe this optimization procedure, as well as a lower bound, in Section 2. We then propose a heuristic procedure based on the iterated local search (ILS) methodology (Section 3). ILS has strong connections with the strategic oscillation originally proposed within tabu search [8]. Specifically, we propose two solution construction procedures (Section 3.1), two mechanisms to improve solutions via neighborhood searches (Section 3.2), and one perturbation strategy. Section 4 describes the computational experiments performed, and finally Section 5 presents the associated conclusions.

2. Branch-and-bound for the BQP

Branch and bound [13] is the process of systematically and exhaustively exploring the solution space by means of a search tree. See Wolsey [20] and Nemhauser and Wolsey [18] for classical references of this search strategy. Recent successful applications can be found in Martí et al. [17] and Martí et al. [16]. The search operates with bounds (lower and upper) on the optimal value of the objective function, a tree structure and exploration strategy. We obtain an initial lower bound (LB) with the heuristic procedures described in the following sections. This lower bound might change in the course of the B&B search and is used alongside an upper bound calculation in order to eliminate (i.e., prune) branches from further consideration.

We utilize a binary tree, where each node is associated with a vertex $i \in I$ in the graph. Each branch represents the decision of whether or not the vertex is included in the solution. Therefore, there are only two branches originating from each node in the tree, including the root node. At each node of the tree (except the root node), there is a set of vertices in I that have been selected, denoted by I'. We point out that it is not necessary to branch on vertices in J because given a selection of vertices in I the corresponding optimal subset of vertices in J can be trivially determined. This optimal selection of the vertices in J (denoted by J') generates a lower bound. In particular, a vertex $j \in J$ is selected if and only if

$$g(j) = c_j + \sum_{i \in I'} q_{ij} > 0$$

The set $S = I' \cup J'$ is a complete solution of the problem and therefore $f(S)$ can be used to update LB when $f(S) > LB$. While a single (the best) lower bound is maintained throughout the search, an upper bound (UB) is associated with each node. The upper bound is used to determine whether additional exploration rooted at the node is warranted. In particular, if for a given node it is determined that $UB \leq LB$, then there is no hope of finding a better solution by completing the sub-tree that is rooted at that node. Corresponding to each node of the tree, there is the set of selected vertices (i.e., I') and also the set I^U of unexplored vertices. The vertices $i \in I$ that belong to neither I' nor I^U are those that have been excluded from the solution by previous branching decisions. An upper bound associated with a node represented by (I', I^U) may be calculated as follows:

$$UB(I', I^U) = \sum_{i \in I'} c_i + \sum_{i \in I^U} \max(0, c_i) + \sum_{j \in J} \max\left(0, g(j) + \sum_{i \in I^U} \max(0, q_{ij}) + \sum_{i \in I'} q_{ij}\right)$$

The upper bound calculation is based on adding all the known weights (i.e., the weights associated with vertices that have been selected) and the strictly positive weights of the vertices that have not been explored. Then, we add the potential weight contribution of each vertex $j \in J$. Only strictly positive contributions are added to the upper bound calculation.

There are two standard techniques to explore a B&B tree: breadth-first and depth-first. Breadth-first generates wide trees and has demanding memory requirements. In most cases, B&B searches cannot be solely conducted on the basis of a breadth-first strategy due to computer memory limits. A depth-first approach attempts to find a leaf as fast as possible before moving to a different node. The memory requirements for depth-first are modest but the effectiveness of the approach is somewhat limited. The best B&B implementations use a mixed strategy that combines both approaches. In our mixed approach, we start with a breadth-first search until memory is exhausted, at which point, we switch to a depth-first exploration. Regardless of the exploring strategy, the direction $i \notin I'$ is always explored first.

3. Iterated local search for the BPQ

Iterated local search [14], usually referred to as ILS, is a metaheuristic based on a modification of local search or hill climbing methods for solving discrete optimization problems. Duarte et al. [3] and Lozano et al. [15] describe successful applications of this methodology. Algorithm 1 summarizes the ILS framework. An initial solution S_0 is generated that is immediately subjected to an improvement procedure (*LocalSearch*). The improved solution S_* becomes the starting point of the main ILS loop. This iterative loop consists of three main functions: *Perturb*, *LocalSearch* and *Accept*. *Perturb* typically employs random elements to change the current solution S_* to produce the perturbed solution S'. *LocalSearch* then attempts to find an improved solution S'_* and *Accept* implements the criteria by which the next current solution S_* is chosen. Both *Perturb* and *Accept* may use recorded history of the search to implement their strategies. For instance, it is possible to use frequency memory à la tabu search in order to bias perturbation mechanisms and acceptance criteria [4,5].

Although not explicit in Algorithm 1, the procedure keeps track of the best solution and returns it upon termination. The criteria within the *Accept* function create a balance between diversification and intensification. The criterion that encourages the most diversification is the one that always accepts S'_* and makes this solution the current solution (i.e.,

$S_* \leftarrow S'_*$). On the other hand, a low diversification criterion is such that S'_* is accepted only if it improves upon the current solution, i.e., only if $f(S'_*) > f(S_*)$. One way of using search history is by accepting a non-improving solution after a number of iterations without improving.

Algorithm 1. Iterated local search

```
Generate initial solution S_0
S_* ← LocalSearch(S_0)
do
  S' ← Perturb(S_*, history)
  S'_* ← LocalSearch(S')
  S_* ← Accept(S_*, S'_*, history)
until termination criteria met
```

Our implementation employs the maximum diversity criterion of always accepting S'_*. We now describe the variants that we have created for constructing initial solutions, perturbing current solutions and locally searching for improved solutions.

3.1. Procedures to generate an initial solution

Our first procedure (C1) to generate an initial solution is based on a semi-greedy strategy that generates a pre-specified number of solutions (k) and chooses the best as S_0. The procedure uses a candidate list of vertices that consists of all vertices that have not been selected (U). Initially, U consists of all vertices in the graph, i.e., $U \leftarrow V$. The current partial construction is given by the set S of vertices that have been selected. This set is initially empty, i.e., $S \leftarrow \emptyset$. Decisions to include vertices in the solution are made on the basis of the "weight potential" of each vertex. The weight potential for a left-hand-side vertex $i \in U$, denoted by $w_p(i)$, is defined as follows.

$$w_p(i) = c_i + \sum_{j \in S} q_{ij} + \sum_{j \in U} \max(0, q_{ij})$$

A similar expression can be constructed for a right-hand-side vertex $j \in U$. The weight potential of a vertex consists of its weight, the sum of the weights of the edges that connect to vertices that have been already selected, and the positive weights of edges that connect to vertices that could be selected in future steps.

Instead of using the pure-greedy approach of selecting the vertex with the highest evaluation (i.e., the one with the largest w_p value), we implement the semi-greedy criterion of randomly selecting a vertex from a restricted list of top candidates. The top candidates are those whose w_p-value is within α% of the maximum w_p value. For instance, if the vertex with the highest potential has a w_p value of 100 and α is set to 10, then only those vertices whose w_p values are at least 90 are allowed to be in the restricted candidate list. This strategy was originally developed by Hart and Shogan [10].

Suppose that at a given step a right-hand-side vertex $j \in U$ is selected, then $S \leftarrow S \cup \{j\}$ and $U \leftarrow U \setminus \{j\}$ and the following expression can be used to update all the weight potentials for the left-hand-side vertices $i \in U$

$$w_p(i) = w_p(i) - \max(0, q_{ij}) + q_{ij}$$

There is an equivalent expression to update right-hand-side vertices $j \in U$ after the selection of a left-hand-side vertex $i \in U$. The procedure stops when all weight potentials of the unselected vertices are strictly negative, that is, when $w_p(v) < 0$ for all $v \in U$. Algorithm 2 summarizes this solution-construction procedure.

Algorithm 2. Construction procedure C1

```
S ← ∅ and U ← V
w_p ← WeightPotential(c, q)
do
  w_max ← max_{v ∈ U} w_p(v)
  RCL ← {v ∈ U : w_p(v) ≥ w_max × (1 − α)%}
  u ← RandomSelection(RCL)
  S ← S ∪ {u} and U ← U \ {u}
  w_p ← UpdateWeightPotential(w_p, u, q)
until w_p(v) < 0 for all v ∈ U
```

Our second procedure (C2) is purely greedy and produces a single solution with minimal computational effort. The procedure is divided into two phases. In the first phase it selects vertices from the left hand side of the graph and in the second phase it selects vertices from the right hand side of the graph. The selection of a vertex $i \in I$ in the first phase depends on the weight potential of the vertex (i.e., $w_p(i)$) as defined above. Since no vertices in J have been selected in this phase of the procedure, the selection function takes on the same form as the one used at the start of C1 (i.e., $w_p(i) = c_i + \sum_{j \in J} \max(0, q_{ij})$). Only those vertices $i \in I$ with $w_p(i) > 0$ are added to the solution S. The first phase finishes when all vertices $i \in I$ have been considered.

In the second phase, the right-hand-side vertices $j \in J$ are chosen. Instead of using the potential weight of the vertices, the actual weights are used as the selection criterion. So, for this phase, we define $w(j)$ as the true weight of vertex $j \in J$

$$w(j) = c_j + \sum_{i \in S} q_{ij}$$

Here again, only those vertices $j \in J$ with $w(j) > 0$ are added to the solution. The phase terminates when all vertices have been considered. In both phases, the list of vertices is scanned in lexicographical order. There is no need to consider any other order given that neither $w_p(i)$ nor $w(j)$ changes due to previous selection decisions within the same phase. Algorithm 3 summarizes this construction procedure.

Algorithm 3. Construction procedure C2

```
S ← ∅
for each (i ∈ I) do
  w_p(i) ← c_i + Σ_{j ∈ J} max(0, q_ij)
  if w_p(i) > 0 then S ← S ∪ {i}
end for
for each (j ∈ J) do
  w(j) = c_j + Σ_{i ∈ S} q_ij
  if w(j) > 0 then S ← S ∪ {j}
end for
```

Due to the symmetry of the q values, we have found through experimentation that it makes no difference in performance to switch the phases and construct the right hand side first followed by the left hand side of the graph.

3.2. Improvement and perturbation methods

We developed two solution improvement methods: a local search and a simple intensification-diversification search. The local search (LS) consists of looking for improvements by adding or

deleting a single vertex. To search for a vertex to add or delete, I and J are scanned in an alternating fashion until no improving move is found. Let S once again represent the set of vertices that have been selected and let $U = V \setminus S$ be the set of vertices that have not been selected. The local search first scans I to identify a vertex $i \in U$ ($i \in S$) that if added to (deleted from) the solution it provides a strictly positive contribution to the objective function. When all vertices in I have been evaluated (to be added or deleted) the search moves to the vertices in J where the same logic is followed. If any improvement move is performed, the scanning of both sets is repeated. The search stops when no improvement move is identified in a single pass through both sets. The method is summarized in Algorithm 4.

Algorithm 4. Local search (LS)

do
  for each $i \in I$ **do**
    $w(i) = c_i + \sum_{j \in S} q_{ij}$
    if $w(i) > 0$ **and** $i \in U$ **then** $S \leftarrow S \cup \{i\}$
    if $w(i) \leq 0$ **and** $i \in S$ **then** $S \leftarrow S \setminus \{i\}$
  end for
  for each $j \in J$ **do**
    $w(j) = c_j + \sum_{i \in S} q_{ij}$
    if $w(j) > 0$ **and** $i \in U$ **then** $S \leftarrow S \cup \{j\}$
    if $w(j) \leq 0$ **and** $i \in S$ **then** $S \leftarrow S \setminus \{j\}$
  end for
until S does not change

Our second improvement method consists of a simple intensification-diversification search (IDS) that has the goal of reaching more than one local optimum. This procedure uses similar strategies as those developed by Kelly et al. [12] and applied to the quadratic assignment problem. The main idea is to alternate intensification and diversification phases until no improvement is achieved for a predetermined number of iterations (*maxIter*). The search starts from a solution S constructed with either C1 or C2. The intensification phase is applied first and consists of identifying all add and delete moves that improve the objective function value of the current solution. Move values for vertices $i \in I$ are calculated with the following expression

$$MoveValue(i) = \begin{cases} c_i + \sum_{j \in S} q_{ij} & \text{if} \quad i \notin S \\ -\left(c_i + \sum_{j \in S} q_{ij}\right) & \text{if} \quad i \in S \end{cases}$$

An equivalent expression is used for the move values associated with vertices $j \in J$. Once no improving move is available, the search switches to a diversification step. All vertices are ordered by their descending move values. Then, the first $d = \beta \times |V|$ moves are performed, with $0 \leq \beta \leq 1$. The β parameter controls the percentage of vertices that will be involved in the search. Then, if β takes value close to 0, just a few vertices would be explored. On the other hand, if β takes values close to 1, most of the vertices would be explored.

The first d vertices in the ordered list are either removed if they are in the solution or added if they are not. After this step is performed, the intensification phase is applied again if no more than *maxIter* iterations have been performed without finding an improved solution. Algorithm 5 summarizes this search.

Algorithm 5. Intensification-diversification search (IDS)

$iter \leftarrow 0$
$d = \beta \times |V|$
while $iter < maxIter$ **do**
  $v^* \leftarrow \operatorname{argmax}_{v \in V} MoveValue(v)$
  if $MoveValue(v^*) > 0$ **then**
    if $v \in U$ **then** $S \leftarrow S \cup \{v\}$
    if $v \in S$ **then** $S \leftarrow S \setminus \{v\}$
    $iter \leftarrow 0$
  else
    $S \leftarrow diversify(d)$
    $iter \leftarrow iter + 1$
  end if
end while

The perturbation method is the final element of our ILS implementation. A solution is defined by the set of vertices S that have been selected. Therefore, for each solution, there is a set $U = V \setminus S$ of vertices that have not been selected. The perturbation consists of randomly choosing a fraction π of vertices in V and changing their current status. That is, if the chosen vertex is in S then it is deleted from the solution and added to U. On the other hand, if the chosen vertex is not in the solution, then it is added to S and deleted from U.

4. Computational experiments

All procedures described above were coded in Java SE7 and run on an Intel Core i7 2600 CPU (3.4 GHz) and 4 GB RAM. The data for the experiments were taken from the literature. More precisely, we employ the tested generated by Karapetyan and Punnen [11]. All the details on how the data instances were generated are found in Section 4 of Karapetyan and Punnen [11]. The testbed[2] contains the following five graph types: random, max biclique, max induced subgraph, maxcut, and matrix factorization.

For each graph type, we consider 17 instances for a total of 85. The 17 instances are divided into three groups

- 7 small instances ($m = \{20, 25, 30, 35, 40, 45, 50\}$ and $n = 50$)
- medium instances ($m = \{200, 400, 600, 800, 1000\}$ and $n = 1000$)
- large instances ($m = \{1000, 2000, 3000, 4000, 5000\}$ and $n = 5000$)

In our first experiment we test our B&B implementation on the 35 small instances. The B&B is compared with the application of Cplex 12.5.1 (with default parameter settings) to the mixed-integer program formulation of the BQP presented in Section 5 of Karapetyan and Punnen [11]. There is also an exhaustive enumeration suggested by Karapetyan and Punnen [11] that we do not include in this comparison because experiments by these authors have shown that this approach is less effective than solving the MIP formulation. Table 1 summarizes the outcome of our experiment. Both procedures were limited to 1800 CPU seconds. We report, average values for the best bounds, the GAP ($UB/LB - 1$), and CPU time associated with each procedure. The CPU time only includes the instances in which the procedures

[2] Available at http://www.cs.nott.ac.uk/~dxk/.

Table 1
Summary of results for B&B and Cplex on 35 small instances.

Metric	B&B	Cplex
LB	14,483.94	14,184.63
UB	23,966.80	17,297.45
GAP	0.79	0.28
CPU seconds	17.50	10.89
No. of optima	21	17

Table 2
Performance measures for construction procedures applied to medium instances.

Method	Avg. obj.	Dev. (%)	No. best	CPU seconds
C1(0.25)	1093,578.04	32.32	4	6.35
C1(0. 50)	**1315,404.08**	11.36	13	5.87
C1(0.75)	1277,685.32	**6.98**	17	5.36
C2	531,586.96	18.29	**20**	**0.01**

terminate before the time limit. We also report the number of optimal solutions verified by each approach.

The B&B is capable of verifying more optimal solutions (21) than Cplex (17) within the time limit. Since the average CPU times shown in Table 1 correspond to the runs in which the procedures terminate before the 1800-s time limit, it can be said that, when compared to the time limit, both procedures find optimal solutions in approximately the same amount of time; however, our B&B is able to find more optimal solutions than Cplex. For the problems for which the optimal solutions could not be verified, we observe that in general Cplex produces tighter upper bounds but lower bounds of slightly inferior quality.

We now turn our attention to the heuristic procedures in the ILS framework. To test these elements, we employ the 25 medium instances. We start by testing the construction methods, C1(α) and C2 (see Table 2). Our performance metrics are: average objective function value (Avg. Obj.), average deviation from the best solution (Avg. Dev.), number of best solutions (#Best) and average CPU time in seconds (CPU seconds). The best solution for each instance is the one identified by all the runs performed within this experiment. That is, this is not necessarily the best-known solution to the problem instance.

The minimum average deviation is achieved by C1 with $\alpha = 0.75$. However, C2 contributes with more "best solutions" than any of the C1 variants. Upon closer examination of the raw results, we have been able to determine that the large deviation associated with C2 is almost entirely due to its poor performance on max biclique instances. When ignoring those results, it becomes clear that C2 outperforms all C1 variants. We keep both C1(0.75) and C2 for further experimentation.

We next test IDS to determine effective values for β and *maxIter*. An experiment using IDS with the a priori default values of $\beta = 0.5$ and *maxIter* = 20 showed that this procedure dominates LS. Therefore, we abandoned further consideration of LS and focused on improving the performance of IDS through parameter tuning. To avoid confounding the effect of the quality of a starting solution and the effectiveness of the IDS, we use random starts for this experiment. We test $\beta = \{0.25,\ 0.50,\ 0.75\}$ and *maxIter* = {10, 20, 30}, and calculate the same performance measures as before. Table 3 summarizes the results of this experiment.

Examining the results in Table 3, it seems reasonable to conclude that *maxIter* should be set to 30. The competing alternatives for the value of β are 0.25 and 0.75. We use as tiebreaker the computing time and elect to choose $\beta = 0.25$ for additional experimentation.

Table 3
Summary of results for several IDS parameter values.

β	*maxIter*	Avg. obj.	Dev. (%)	No. best	CPU seconds
0.25	10	1307,970.12	8.57	5	**0.29**
	20	1309,651.76	8.41	14	0.57
	30	1310,071.60	8.39	**15**	0.75
0.50	10	957,874.64	11.77	3	0.92
	20	960,189.76	11.49	5	1.81
	30	960,462.16	11.45	5	2.40
0.75	10	1505,916.48	3.49	4	1.81
	20	1506,109.52	3.46	4	2.85
	30	**1507,039.32**	**3.34**	5	4.85

Table 4
Summary of results for ILS with two different starting solutions.

Construction	π	Avg. obj.	Dev. (%)	No. best	CPU seconds
C1(0.75)	0.10	1753,657.20	0.72	**10**	47.01
	0.20	1745,306.32	0.71	3	50.43
	0.30	1712,148.16	1.34	0	59.60
	0.40	1642,497.24	2.66	2	67.95
C2	0.10	**1757,667.40**	**0.27**	4	**40.75**
	0.20	1744,390.40	0.66	9	44.56
	0.30	1697,604.08	1.62	2	55.30
	0.40	1669,465.88	2.20	1	62.61

Table 5
Comparison of two ILS variants with the state-of-the-art.

Method	Avg. obj.	Dev. (%)	No. best	CPU seconds
$V_1^{ex}(R)$	13,369,203.60	3.28	0	**1.71**
$M(V_1^{ex})$	13,873,178.39	1.96	1	327.09
$R_{m/3}^{m}$	**14,420,402.57**	0.65	2	327.09
ILS-C1(0.75)	13,650,410.42	1.29	7	653.71
ILS-C2	14,337,611.18	**0.60**	**13**	160.00

Now that we have identified the most effective configurations of the individual elements of the ILS, we put them all together to calibrate the value of π, that is, the percentage of vertices to peturb in each iteration. We tried $\pi = \{0.10,\ 0.20,\ 0.30,\ 0.40\}$ starting from C1(0.75) and C2 constructions and applying IDS with $\beta = 0.25$. The termination criterion for ILS is set to 100 iterations. The results are summarized in Table 4.

Clearly, small π values perform better than larger ones regardless of the method that is used to construct the initial solution. The results in Table 4, however, do not allow us to make a final decision on whether C1(0.75) or C2 is produces better outcomes in combination with the other search elements. Hence, for our final set of experiments where we compare ILS with the state of the art, we keep two variants ILS-C1(0.75) and ILS-C2 with $\pi = 0.10$.

Karapetyan and Punnen [11] developed 25 heuristic procedures, grouped in 3 categories: Fast heuristics, Portions-based & MultiStart heuristics y Row Merge heuristics. We have chosen the best procedure in each category, namely, $V_1^{ex}(R)$, $M(V_1^{ex})$, and $R_{m/3}^{m}$. Table 5 shows the results of applying these procedures to the 50 instances in the medium and large sets.

We applied Friedman's test to the raw data obtained in the previous experiment. This test ranks each method for each instance in the data set. That is, for each instance, the method that performs the best is assigned the number 1 ranking followed by the second best and so on. Then, an average ranking is

Table 6
Results of Friedman's test.

Method	Rank
ILS-C2	1.11
ILS-C1(0.75)	1.32
$M(V_1^{ex})$	1.60
$R_{m/3}^m$	2.69
$V_1^{ex}(R)$	3.28

Table 7
Results of the sign test.

ILS variant	Benchmark	R^+	R^-	Difference
ILS-C1(0.75)	$V_1^{ex}(R)$	44	6	+
	$M(V_1^{ex})$	35	14	+
	$R_{m/3}^m$	32	18	∼
ILS-C2	$V_1^{ex}(R)$	46	4	+
	$M(V_1^{ex})$	38	12	+
	$R_{m/3}^m$	33	17	+

Table 8
Comparison of two ILS variants with the state-of-the-art.

Method	Avg. obj.	Dev. (%)	No. best
Hybrid	14,575,952.58	0.43	34
ILS-C2	14,455,832.30	0.26	22

calculated for each method. A small p-value associated with this test indicates that the averages are indeed significantly different. We obtained a p-value of 0.000 for the data that we used to produce Table 5. The ranking is shown in Table 6.

We apply the sign test to compare the ILS variants with the three best previously proposed procedures. We use a p-value of 0.05 to detect significant statistical difference in this test. Table 7 summarizes the results of this test, where R^+ indicates the number of times that the ILS variant produced a better outcome than the procedure being compared, R^- represents the number of times when the benchmark procedure was better than the ILS variant, and "Difference" indicates whether there is a positive significant difference (+), a negative significant difference (−) or no significant difference (∼). A positive significant difference means that results obtained with the ILS variant are significantly better than the benchmark procedure. The opposite is true for negative significant differences.

The results of the sign test are such that in all but one case there is a significant positive difference in favor of the ILS variants. Although ILS-C1(0.75) obtains better results than $R_{m/3}^m$ in 32 out of 50 instances, the associated p-value for that test is larger than 0.05 and therefore the difference is not considered statistically significant. One more instance would have made the difference, as it did in the case of ILS-C2 vs. $R_{m/3}^m$.

In our final experiment, we compare our method ILS-C2 with the best heuristic in Glover et al. [9] called *Hybrid*. In particular, this method combines a tabu search algorithm with a very-large scale neighborhood search. We thank Professors Punnen and Ye for providing us with their executable code to perform a fair comparison between our algorithm and *Hybrid*. Table 8 shows the results of applying both procedures to the 50 instances in the medium and large sets. Both algorithms are executed for 1000 s.

Results in Table 8 clearly show that both methods obtain high quality solutions; however, it is difficult to disclose which one is the best. On the one hand, our method exhibits better performance when considering the average deviation. In particular, ILS-C2 achieves a remarkable 0.26% while the *Hybrid* method obtains 0.43%. On the other hand, *Hybrid* matches a significant larger number of best solutions than ILS-C2 (34 vs. 22). In order to determine the best procedure, we additionally conduct a sign test to evaluate whether a procedure outperforms the other or not. As it is customary, we consider a probability threshold of 0.05 to detect significant statistical differences in this test. The associated p-value of 0.10 (considerably larger than 0.05), indicates that there are not significant differences between both procedures. Therefore, it is not possible to determine the superiority of one method over the other one.

5. Conclusions

We have used the BQP as the context to explore some ideas in both exact and heuristic search. Within the framework of B&B, we were able to determine that fairly straightforward bounds coupled with a mixed exploration strategy was sufficient to find optimal solutions to problem instances for which these solutions were not available. Heuristically, we select the Iterated Local Search framework to test several configurations resulting from combining construction and neighborhood search procedures. The resulting procedure creates new benchmarks and establishes a new state-of-the-art for BQP. Our experiments show that a carefully balance neighborhood search that is allowed to move beyond the first local optimal point is a strategy that is effective regardless of the initial solution. Our overall goal for this project was to combine search elements that have been identified as the core contributors to well-known methods. For instance, the semi-greedy constructions are at the core of GRASP [6] iterations while diversification-intensification strategies based on memory information are the essence of tabu search. Our approach – of identifying and combining key search components – is in sharp contrast with the idea of creating procedures by forcing some sort of analogy to biological of physical systems, something that has sadly become a popular trend.

Acknowledgments

This research has been partially supported by the Ministerio de Educación Cultura y Deporte of Spain (Grant Ref. PRX12/00016), the Ministerio de Economía y Competitividad of Spain (Grant Ref. TIN2012-35632-C02) and the Generalitat Valenciana (Prometeo 2013/049).

References

[1] Alon N, Naor A. Approximating the cut-norm via Grothendieck's inequality. SIAM J Comput 2006;35:787–803.
[2] Ambühl C, Mastrolilli M, Svensson O. Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. SIAM J Comput 2011;40:567–96.
[3] Duarte A, Martí R, Álavarez A, Ángel-Bello F. Metaheuristics for the linear ordering problem with cumulative costs. Eur J Oper Res 2012;216:270–7.
[4] Duarte A, Glover F, Martí R, Gortázar F. Hybrid scatter tabu search for unconstrained global optimization. Ann Oper Res 2011;183:95–123.
[5] Duarte A, Laguna M, Martí R. Tabu search for the linear ordering problem with cumulative costs. Comput Optim Appl 2011;48:697–715.
[6] Feo TA, Resende MGC. Greedy randomized adaptive search procedures. J Global Optim 1995;6(2):109–33.
[7] Gillis N, Glineur F. Low-rank matrix approximation with weights or missing data is NP-hard. SIAM J Matrix Anal Appl 2011;32:1149–65.
[8] Glover F, Laguna M. Tabu search. New York: Springer; 1997 (978-0-7923-8187-7).

[9] Glover, F., T. Ye, A.P. Punnen, G. Kochenberger (2014) Integrating tabu search and VLSN search to develop enhanced algorithms: a case study using bipartite boolean quadratic programs, ⟨arXiv:1305.5610⟩ [cs.AI].
[10] Hart JP, Shogan AW. Semi-greedy heuristics: an empirical study. Oper Res Lett 1987;6(3):107–14.
[11] Karapetyan, D., A. Punnen (2013) Heuristic algorithms for the bipartite unconstrained 0-1 quadratic programming problem ⟨arXiv:1210.3684v2⟩ [cs.DM].
[12] Kelly JP, Laguna M, Glover F. A study on diversification strategies for the quadratic assignment problem. Comput Oper Res 1994;21(8):885–93.
[13] Land AH, Doig AG. An automatic method of solving discrete programming problems. Econometrica 1960;28(3):497–520.
[14] Lourenço, H.R., O. Martin, Stützle T. (2010). Iterated local search: framework and applications". Handbook of metaheuristics, 2nd ed., International series in operations research & management science, Vol. 146. Kluwer Academic Publishers, p. 363–397.
[15] Lozano M, J., Duarte A, Gortázar F, Martí R. A hybrid metaheuristic for the cyclic antibandwidth problem. Knowl Based Syst 2013;54:103–13.
[16] Martí R, Pantrigo JJ, Duarte A, Pardo EG. A branch and bound algorithm for the cutwidth problem. Comput Oper Res 2013;40(1):137–49.
[17] Martí R, Gallego M, Duarte A. A branch and bound algorithm for the maximum diversity problem. Eur J Oper Res 2010;200:36–44.
[18] Nemhauser GL, Wolsey LA. Integer and combinatorial optimization. Wiley Interscience Series in Discrete Mathematics and Optimization; 1988.
[19] Punnen, A.P., P. Sripratak, D. Karapetyan (2012) The bipartite unconstrained 0-1 quadratic programming problem: polynomially solvable cases ⟨arXiv:1212.3736v1⟩ [math.OC].
[20] Wolsey LA. Heuristic analysis, linear programming and branch and bound. Comb Optim II: Math Program Stud 1980;13:121–34.